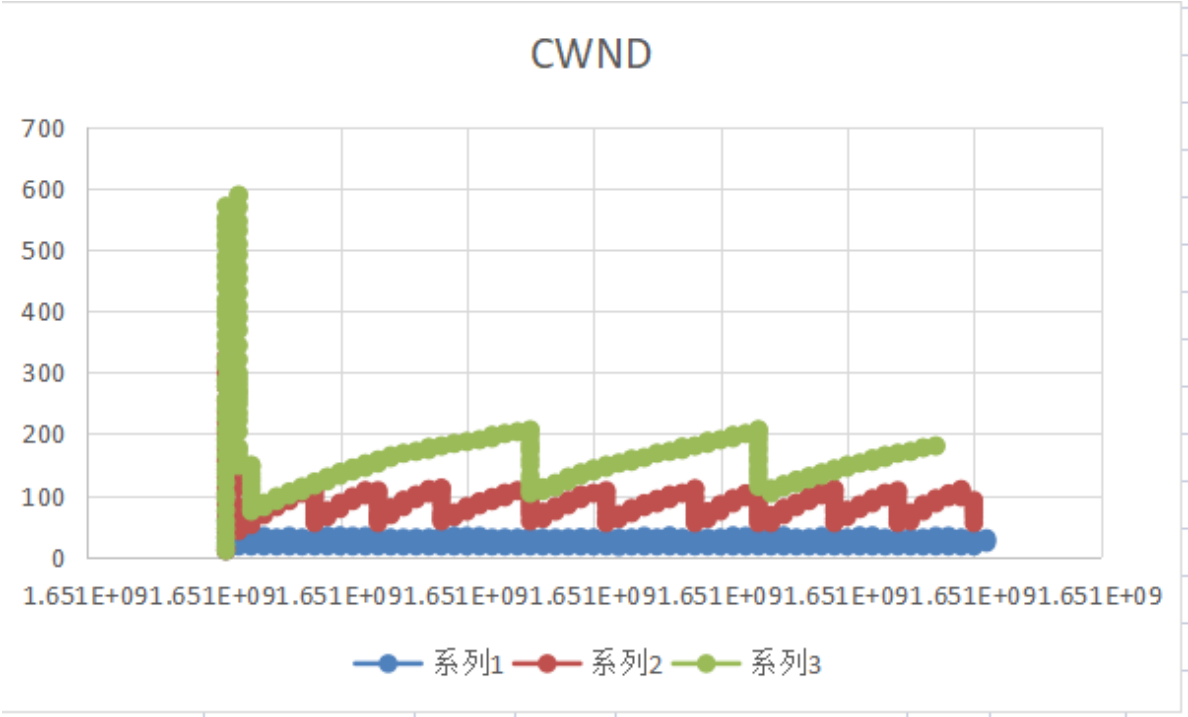
数据包队列管理实验

重现Bufferbloat问题

CWND

取qlen长度为10，50，100。利用excel画图得到CWND的bufferbloat现象

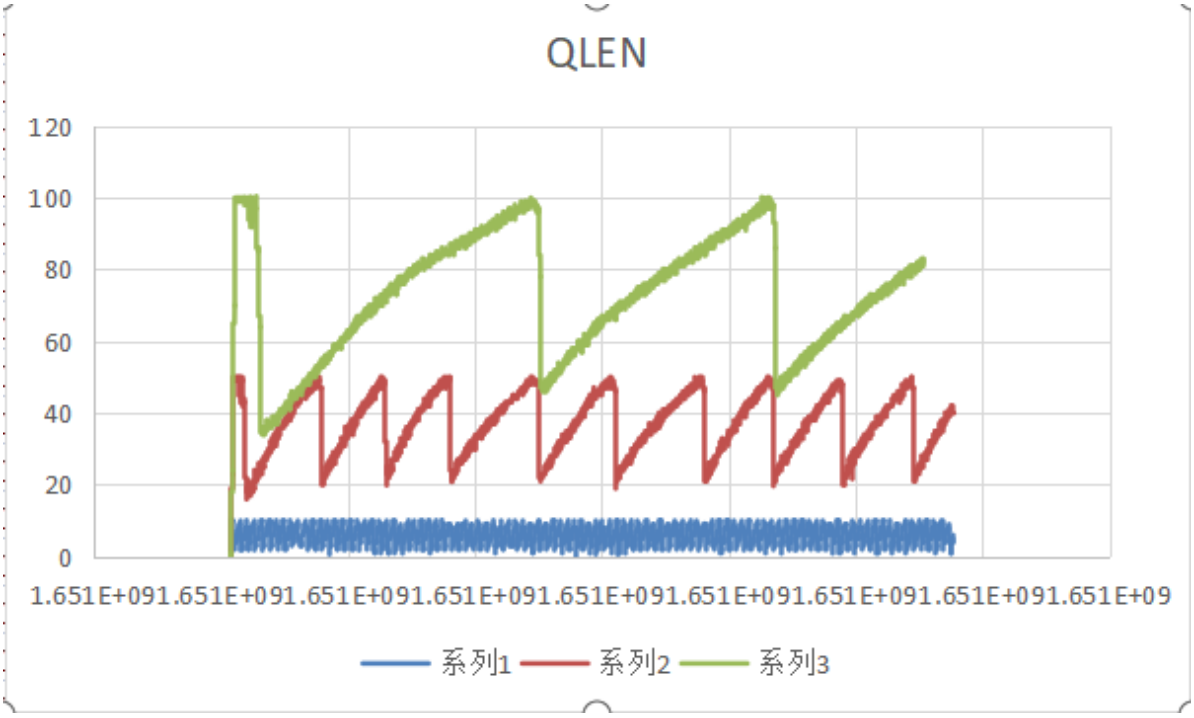


其中蓝色表示qlen=10,红色表示qlen=50,绿色表示qlen=100

迅速增长到最大值，发现丢包后，再迅速回落到原本的队列大小。队列越小，波动越明显。

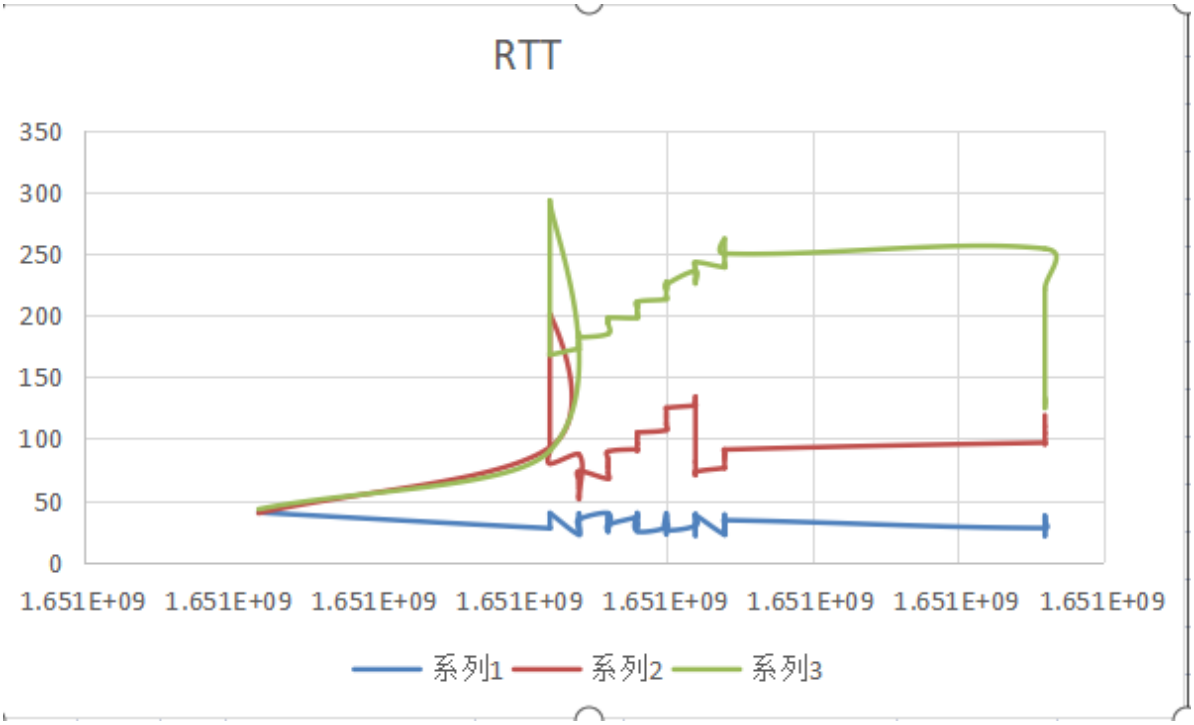
QLEN

取qlen长度为10，50，100。其中蓝色表示qlen=10,红色表示qlen=50,绿色表示qlen=100



同样观察到，迅速增加后，发现丢包，则降低到原先的50%左右。原因在于，只有观察到了拥塞，才发现拥塞现象。同样地，qlen长度越小，波动越频繁。

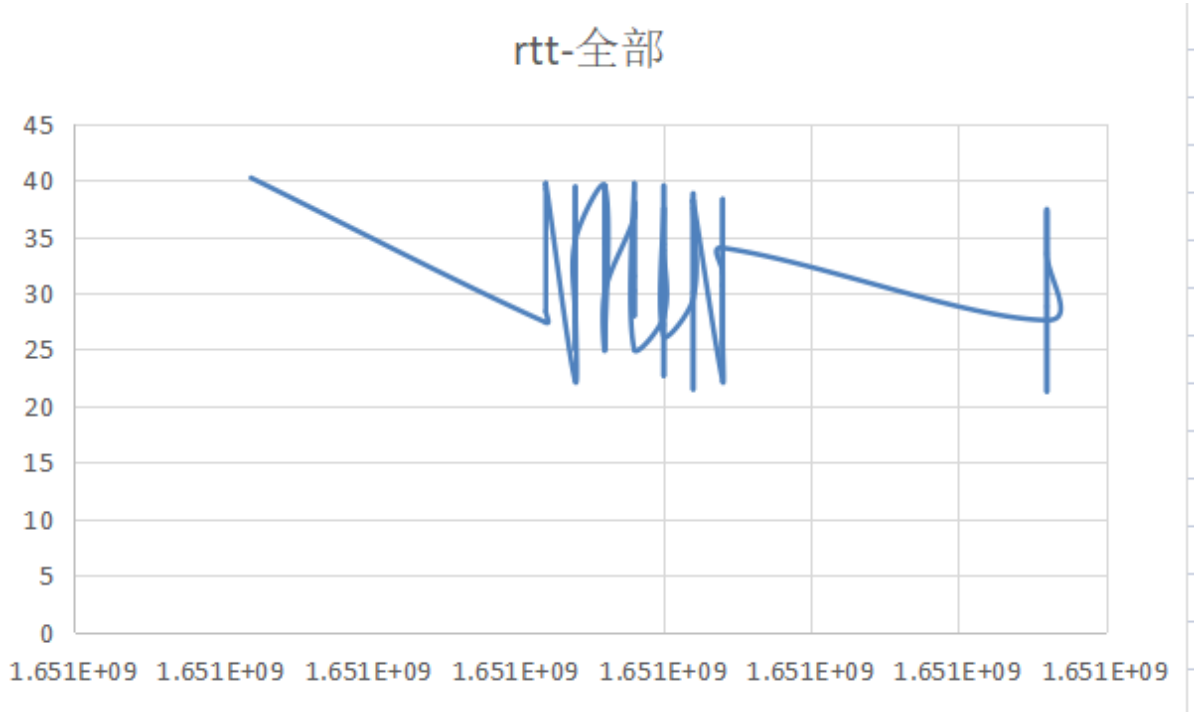
RTT



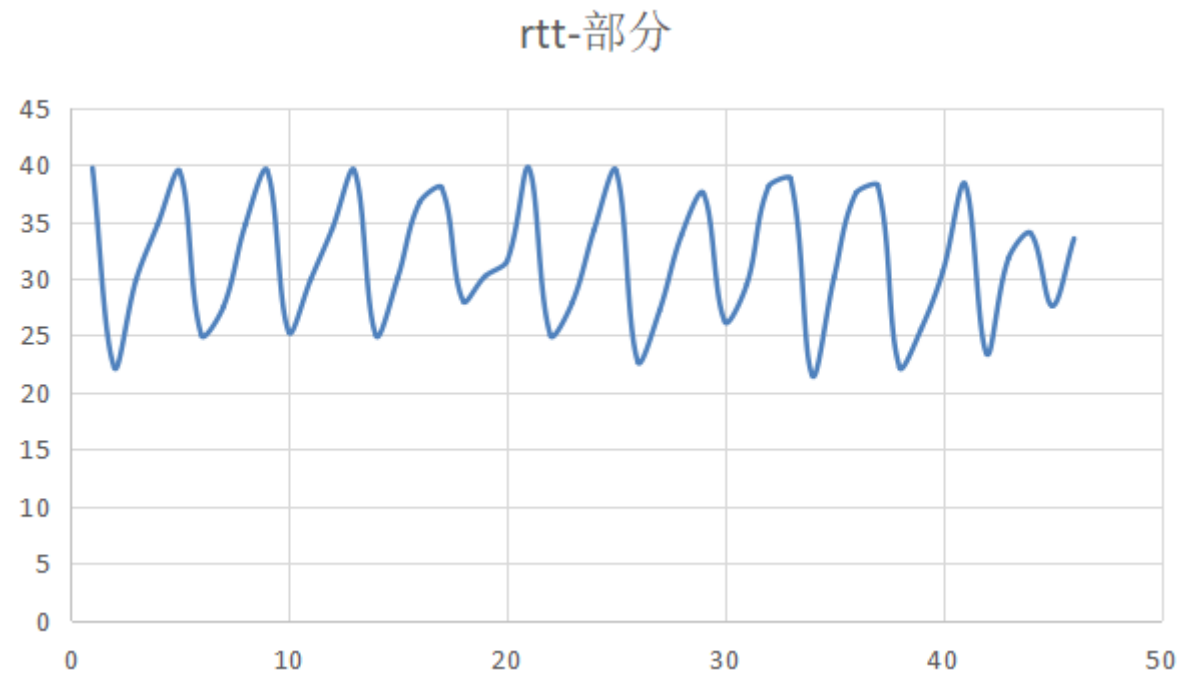
这里坐标需要进行一些调整，蓝色表示qlen=10,红色表示qlen=50,绿色表示qlen=100。

观察到，波动频率与队列的长度密切相关。上图由于数据点过多，所以并不算直观，对单个图像的数据进行局部的采集，现象更加明显。

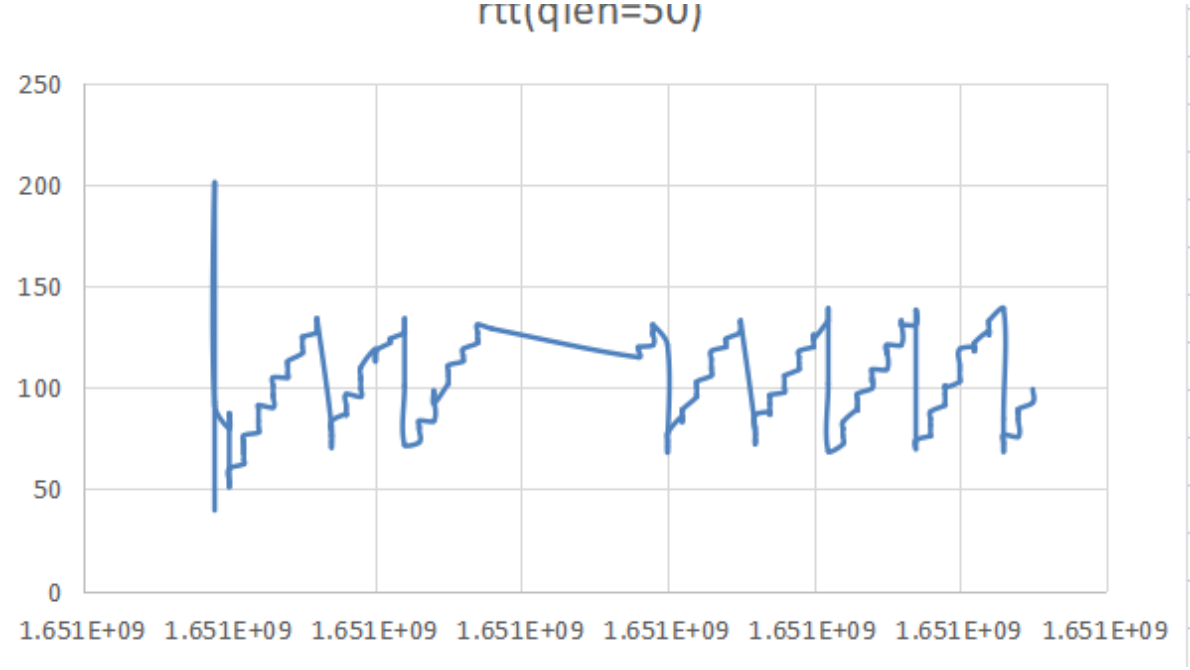
当对全部的rtt数据采样时，图像如下（qlen=10）



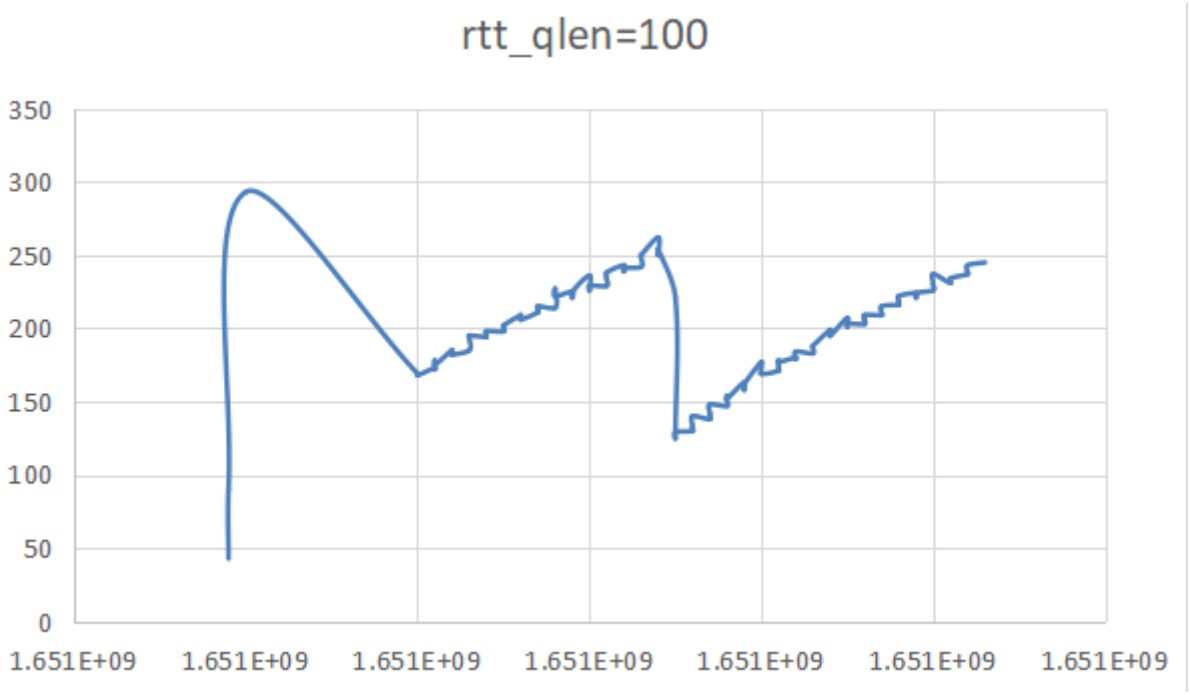
取局部数据，则得到更加合适的图像复现



qlen=50



qlen=100



调研思考题

1.新型拥塞控制机制(BBR)如何解决Bufferbloat问题?

解：首先改变了互联网广泛使用基于丢包的拥塞控制算法，传统的算法认为丢包和拥塞等效。这不一定绝对正确，因为当网络交换机和路由器的缓冲区适配当时的网络带宽时，可以认为丢包等于拥塞，但当发送者速率足够快时，缓冲区会被快速填满，从而引发丢包。

拥塞的根本原因是**在网络路径中，传输中的数据量始终大于时延-带宽积。**而在非拥塞情况下，也会频繁出现。

BBR算法直接对网络建模来避免真实的拥塞。

BBR使用状态机来维护三个控制参数，状态机交替循环探测 BBR.BtlBw(估算的传输通道瓶颈带宽，

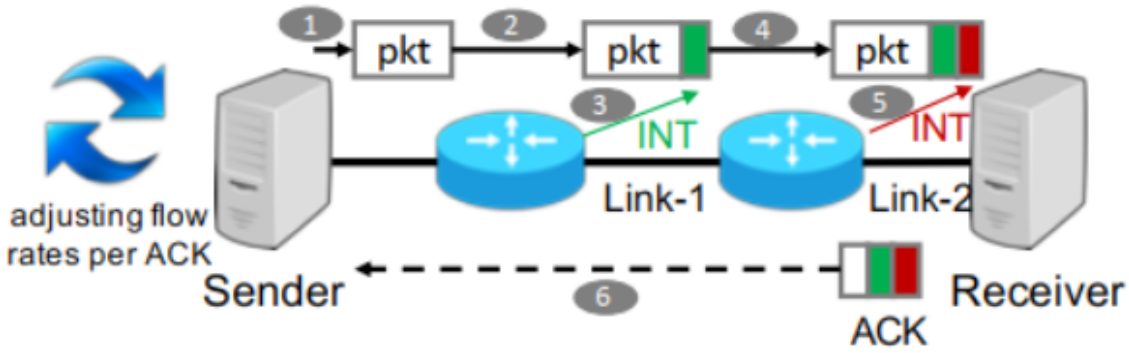
源自滑动窗口的最大投递速率样本)。BBR.RTprop(估算的该路径的双向往返传播延时，源自滑动窗口的最小往返延时样本。)，用于实现高吞吐量，低延时，近似公平的带宽共享。

BBR起始于startup状态，该状态下迅速提升发送速率。当预估到网络管道被填满时，则进入Drain状态，开始排放管道队列。已经处于稳定状态的BBR流将只使用ProbeBW状态，去周期性地短暂提升传输中的数据量，来探测更高的BBR.BtlBw样本。ProbeRTT状态下，会短暂地降低传输中的数据量，去探测更低的BBR.RTprop样本。

2.新型拥塞控制机制（HPCC）如何解决Bufferbloat问题?

解：在传统的CC方案中，存在对于粗粒度反馈信号，例如RTT,收敛慢，不可避免的数据包排队，以及复杂的参数调整问题。

下图为HPCC的框架图：



HPCC是一个发送方驱动的CC框架。发送方发送的每一个数据包都将被接收方确认。在包从发送方传播到接收方的过程中，沿路径的每个交换机利用其交换ASIC的INT特性来插入一些元数据，这些元数据报告包的出口端口的当前负载，包括时间戳(ts)、队列长度(qlen)、传输字节(txBytes)和链路带宽容量(B)。

当接收方收到数据包时，它将交换机记录的所有元数据复制到它发送回发送方的ACK消息中。发送方决定如何在每次收到带有网络负载信息的ACK时调整其流量速率。

HPCC是一种基于窗口的CC方案，用于控制inflight字节数。inflight字节表示已经发送但尚未在发送方确认的数据量。inflight字节直接对应于链路利用率。对于一个链路，Inflight字节是所有通过它的流的总inflight字节。假设一条链路的带宽为B,穿过它的第i个流的窗口大小为Wi。此链路的inflight字节为所有的Wi的总和。需要控制每个链路的I = sum{Wi} < B × T，B是带宽，T是基本传播RTT。

HPCC需要进行多轮调整故障窗口，来解决拥塞问题。

具体的算法如下：

Algorithm 1 Sender algorithm. $ack.L$ is an array of link feedbacks in the ACK; each link $ack.L[i]$ has four fields: $qlen$, $txBytes$, ts, and B. L is the sender's record of link feedbacks at the last update.

1: **function** MEASUREINFLIGHT(ack)
2: $u = 0$;
3: **for** each link i on the path **do**
4: $txRate = \frac{ack.L[i].txBytes - L[i].txBytes}{ack.L[i].ts - L[i].ts}$;
5: $u' = \frac{\min(ack.L[i].qlen, L[i].qlen)}{ack.L[i].B \cdot T} + \frac{txRate}{ack.L[i].B}$;
6: **if** $u' > u$ **then**
7: $u = u'$; $\tau = ack.L[i].ts - L[i].ts$;
8: $\tau = \min(\tau, T)$;
9: $U = (1 - \frac{\tau}{T}) \cdot U + \frac{\tau}{T} \cdot u$;
10: **return** U;
11: **function** COMPUTEWIND(U, $updateWc$)
12: **if** $U >= \eta$ or $incStage >= maxStage$ **then**
13: $W = \frac{W^c}{U/\eta} + W_{AI}$;
14: **if** $updateWc$ **then**
15: $incStage = 0$; $W^c = W$;
16: **else**
17: $W = W^c + W_{AI}$;
18: **if** $updateWc$ **then**
19: $incStage++$; $W^c = W$;
20: **return** W;
21: **procedure** NEWACK(ack)
22: **if** $ack.seq > lastUpdateSeq$ **then**
23: W = COMPUTEWIND(MEASUREINFLIGHT(ack), $True$);
24: $lastUpdateSeq = snd\_nxt$;
25: **else**
26: W = COMPUTEWIND(MEASUREINFLIGHT(ack), $False$);
27: $R = \frac{W}{T}$; $L = ack.L$;